The Need for Speed: An Analysis of Brazilian Malware Classifiers

Fabrício Ceschin, Felipe Pinagé, Marcos Castilho, David Menotti, Luiz S. Oliveira, André Grégio | Federal University of Parana

Using a dataset containing about 50,000 samples from Brazilian cyberspace, we show that relying solely on conventional machine-learning systems without taking into account the change of the subject's concept decreases the performance of classification, emphasizing the need to update the decision model immediately after concept drift occurs.

The sheer amount of malware variants released daily underscores the urgent need to create automated classifiers/detectors. In general, malware classification through machine-learning techniques is based in the literature on extracting program characteristics in a way that they can be classified as malicious or benign or even grouped into distinct malware families, depending on the attributes and datasets used. Shortcomings of these prior efforts are their lack of reproducibility and the neglect of authors in many cases to provide the data, scripts, or programs and settings for sample classification, analysis, and determining the representativeness of sample selection. Moreover, the focus of most prior work has been on obtaining accuracy rates close to 100 percent, without addressing the challenge of dynamic updating of models as time goes by and new malware samples appear.[1–3] Because there is little discussion about the effectiveness of the classifiers over time and no available data for new tests by other researchers, there are no guarantees that the results by prior machine-learning models are exempt from the bias caused by the chosen dataset.

Digital Object Identifier 10.1109/MSEC.2018.2875369
Date of publication: 21 January 2019

Our work leveraged a set of representative, unfiltered (that is, without preselection of malware families) benign software and malware samples collected in Brazilian cyberspace from 2012 to 2018. Malware attacks in Brazil are strongly motivated by profit. Attackers are intent on either stealing the credentials of people banking via the Internet[4] or they are trying to lure Internet users into paying fake bills.[5] On the one hand, Brazilian's cultural and economic aspects make the malware menace distinct from that in other countries. On the other hand, popular programs worldwide that are considered benign elsewhere tend to be also popular in Brazil.

After the data collection, we extracted the attributes of the samples using attributes from their portable executable (PE) header. With that, we performed traditional experiments in the literature, with the focus on obtaining accuracy rates close to 100 percent, followed by experiments using thresholds that are of particular interest in the industry. Then, we simulated the behavior of conventional learning systems using two classifiers [k-Nearest Neighbors (KNN) and Random Forest], updating their databases over time with known samples to predict new ones. Then, we compared this approach when updating the decision model only after concept drift occurred, using concept-drift detection algorithms, such as the Drift Detection Method (DDM) and the Early Drift Detection Method (EDDM). We also analyzed what happened in the detected drift points on our dataset, using t-distributed stochastic neighbor embedding (t-SNE) projections to show differences in the data related to these points.

Dataset

To create the dataset presented in this article, we used a set of ≈200 GB of executable files (malware and benign software) collected in Brazilian cyberspace or popular Internet download sites from 2012 to early 2018. We collected two classes of software to build the dataset: goodware (allegedly benign software) and malware. To obtain goodware samples, we implemented a web crawler that downloaded software from three sources: Sourceforge, Softonic, and CNET Download. We collected ≈130 GB of binary files, which we assumed benign, totaling 21,116 unique samples. We have an established long-standing partnership with a major Brazilian financial institution (which prefers to remain anonymous) that provides us with daily malware samples collected from detected infections in its corporate perimeter or identified by customers via phishing e-mail attachments. As the malware samples were received by our server, we grouped them by day to save the temporal information. This process has been executed in an ongoing fashion since January 2013, with the exception of the period from January to July 2016, when the collection was suspended due to a shutdown period in the storage server. At the time this article was being written, we had collected approximately 80 GB of malicious binary samples, totaling 29,704, of which 23,033 are unique.

After collecting the data, we extracted as many static attributes as possible from the downloaded executables with the goal of making these available to the research community without direct malware sharing, a practice that is illegal in Brazil. We used Python's pefile library (github.com/erocarrera/pefile), which allowed us to access the attributes of PE files.[6] We extracted 27 attributes from each sample. Of these, 22 are numerical attributes (integer or floating-point numbers, such as base of code, base of data, characteristics, dynamic link library characteristics, file alignment, image base, machine, magic, number of relative virtual addresses and their sizes, number of sections, number of symbols, PE type, pointer-to-symbol table, size, size of code, size of headers, size of image, size of initialized data, size of optional header, size of uninitialized data, time date stamp, and entropy), three textual attributes (set of words, such as list of dynamic libraries, functions, and compilers/tools used), and two unique attributes (unique for each executable MD5 and SHA1 hashes). It is worth noting that only numerical and textual attributes are used in this article: unique attributes are not of interest for machine-learning algorithms, given that they are not discriminant for the classes, only for the files.

Before using these attributes in a machine-learning model, it is necessary to preprocess and normalize them. The preprocess consists of transforming the textual attributes into numerical attributes; that is, the set of words is converted into a set of floating-point numbers that can be "understood" by a classifier. The aim of this preprocessing is to cluster similar texts (documents). This way, programs that use the same libraries, functions, and compilers tend to be close in the features' space. To make that possible, each set of words is transformed in a document (in the context of information retrieval), where each word is separated by a space. Thus, three "documents" are created by file, one for each textual attribute.

Following that, all text is case-folded and normalized: text content was kept in lowercase, and any special characters, accent marks, and numbers were removed. Therefore, we obtained statistical measures about every document based on their words, using the vector space model (VSM), which represents a document through the relative importance of its words.[7] With that, the programming language, compiler, packer, and imported libraries used as well as the functions the program invoked can help to differentiate benign and malicious software. Each text/document is represented by a sparse vector that contains its term frequency–inverse document frequency, a statistic measure used to evaluate how important a word is to a document in relation to a collection of documents,[8] and measures for each word in the vocabulary, which can be reduced to a number V of words that are most present in the given documents or words that are between a range of thresholds (minimum and maximum document frequency).

In this article, we use a range of thresholds (minimum document frequency of 10 percent and maximum of 45 percent) because doing so obtains better results than a fixed vocabulary. Finally, after preprocessing all textual attributes, each VSM vector is concatenated to the other attributes of the file, resulting in a new vector with $22 + 3 \times V$, where 22 is the number of numerical attributes and V is the size of the vocabulary that resulted after applying the range of thresholds. However, there is still a difference in the extracted characteristics scale, leading to the need to normalize them. The normalization technique applied on the extracted features was the MinMax, which scales every feature (characteristic) into an interval between zero and one. From this moment on, all the files are normalized and ready for the classification step.

Experiments

Using the representations (set of features) extracted, we initially performed experiments with four classifiers: multilayer perceptron (MLP), the most frequent type of neural network used for classification purposes; support vector machine (SVM); KNN, a distance-based supervised clustering algorithm; and Random Forest, an ensemble of decision trees. The MLP used is implemented by TFLearn, a high-level application programming interface built on top of TensorFlow, the Google artificial intelligence library.[9] For these experiments, we considered an MLP that consisted of two hidden layers: the first with $F/2$ neurons and the second with $F/3$, where F is the number of features extracted, which varies according to the VSM.

The remaining classifiers used are implementations from the Scikit Learn library,[10] where the chosen SVM (support vector classification implementation) was set to use linear kernel (default $C = 1$)—we did experiments with the radial basis function kernel and grid-search, but the results were not promising—while KNN had the value of K fixed in 5. We tried several other parameter values, but the ones mentioned produced better overall results. To validate our experiments, we used 50 percent of the goodware and malware samples in the testing set, while the remaining were used in the training set. We repeated this procedure 10 times using cross-validation with 10 partitions (folds). Note that this approach is considered traditional in the literature, with the focus on obtaining accuracy rates close to 100 percent.

We calculated the average values obtained in this training/testing process for the measures of accuracy, recall, precision, and F1 score (the harmonic mean of precision and recall)—classic measures for binary classification problems (malware/goodware) where class is imbalanced—for each of the four applied classifiers. The Random Forest classifier achieved the best overall result, with better values for all metrics (98.00 percent for accuracy, 98.07 percent for F1 score, 97.52 percent for recall, and 98.63 percent for precision).

Random Forest is usually successfully applied to other classification problems that rely on security data, such as offensive tweets, malware in general, deanonymization, malicious web pages, and spam. It was followed by the KNN (97.11 percent for accuracy, 97.22 percent for F1 score, 96.91 percent for recall, and 97.54 percent for precision), MLP (96.98 percent for accuracy, 97.11 percent for F1 score, 97.04 percent for recall, and 97.07 percent for precision), and SVM (94.85 percent for accuracy, 94.93 percent for F1 score, 92.39 percent for recall, and 97.60 percent for precision), which was about three percentage points below Random Forest in almost all metrics. However, the presented metrics may not be useful for real applications in this area because false positives can have a direct influence on users' actual protection. This happens because a classifier that causes legitimate software to be blocked/quarantined would still represent a precision of 100 percent. Also, this is not a good approach to evaluate this problem because cross-validation takes into account "samples from future"; that is, data from different epochs are mixed in the training and validation sets, which can help the classifier obtain better results (ignoring the concept-drift problem).

When a user installs a benign program, the running antivirus must be flexible enough to avoid classifying this new software as malware. Hence, the number of false positives should be decreased, even though it means some malware will not be identified. In machine learning, this can be achieved using a threshold that defines a minimum value for an item to be effectively labeled. For this reason, we evaluated the classifiers' performance with distinct thresholds to decrease the number of false positives. In Figure 1, we present the false-positive rate (FPR) and false-negative rate (FNR) for the SVM, MLP, KNN, and Random Forest classifiers, respectively (thresholds range between zero and one). As expected, the higher the threshold, the higher the number of false negatives; that is, the number of undetected malware grows as they are classified as goodware. On the other hand, while the false negatives increase, the false positives decrease, reducing the risk of classifying goodware as malware. Particular observations regarding distinct thresholds T are as follows.

- SVM's FNR starts to increase and its FPR starts to decrease slowly when T is close to 50 percent. When T is close to 95 percent, the FNR starts to increase drastically, achieving a value close to 85 percent when T is near 100 percent, while the FPR achieves a value close to 0 percent.
- MLP's FNR starts to increase slowly when T is close to 50 percent, while its FPR decreases slowly. FNR starts to increase drastically when T is close to 90 percent (value close to 5 percent). However, in this case, the FPR is already near 0 percent, showing a good result from this classifier.
- KNN presents more consistent results. Its FPR starts to decrease from $T = 60$ percent and remains stable up to 80 percent, when the value drops to ≈ 0 percent and remains stable. The opposite can be observed in the

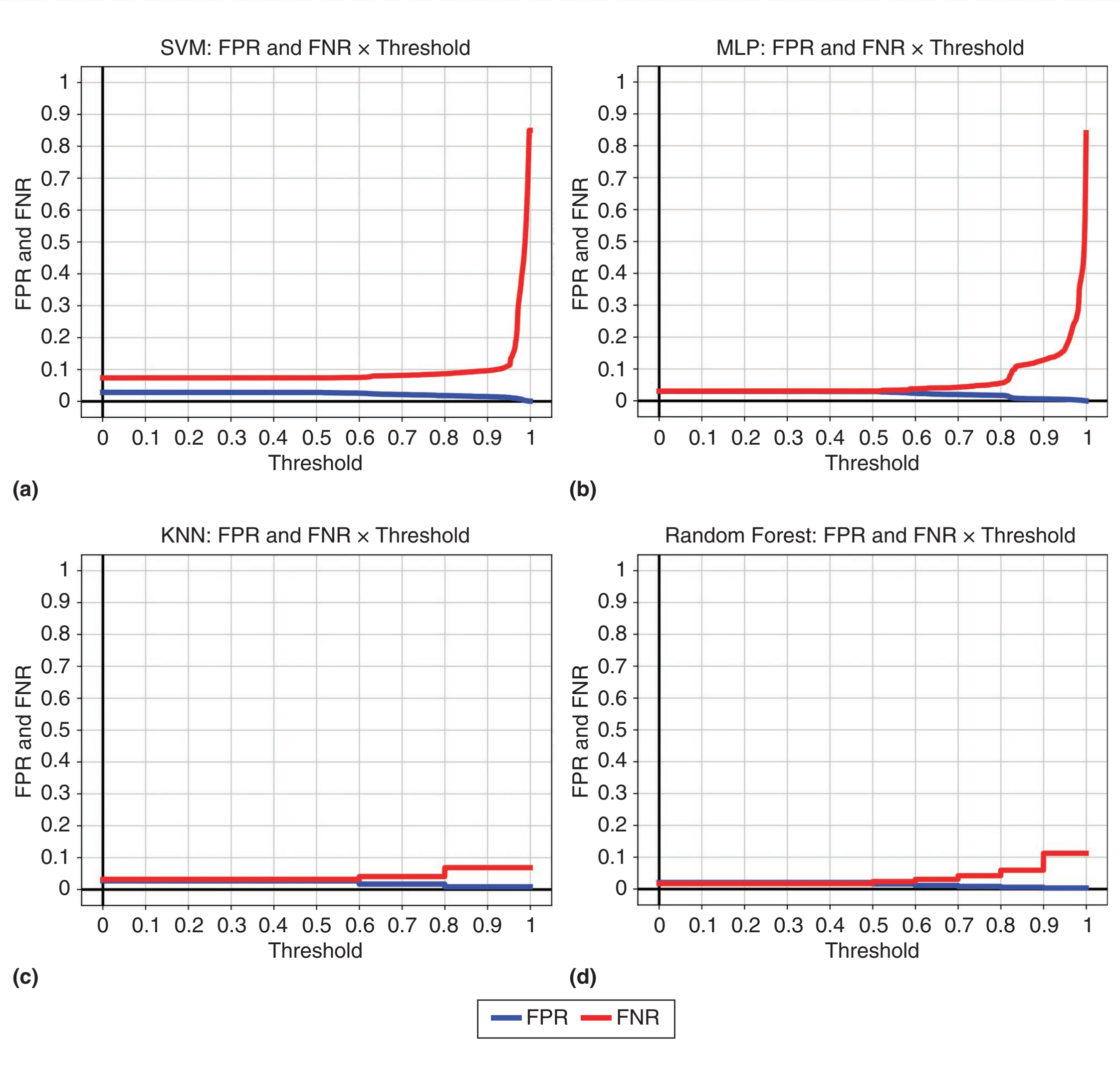

Figure 1. Threshold experiments using (a) SVM, (b) MLP, (c) KNN, and (d) Random Forest classifiers.

FNR: the values keep stable at around 5 percent from $T = 80$ percent on.

- Random Forest seems to have properties even more interesting for this problem; given that at $T = 50$ percent, its FPR and FNR are below 5 percent (really close to 0 percent). However, from $T = 90$ percent on, although the FNR stays close to zero, the FNR reaches almost 10 percent.

In Figure 2, we show the results of the receiver operating characteristic curve of the classifiers. It clarifies what has been previously reported in the figures: due to the threshold increase, the FPR will be lower; that is, fewer samples of goodware will be misclassified. However, the side effect on the true-positive rate (TPR) is that it tends to decrease. The curve also shows the consistency of Random Forest, which maintains a low FPR and still has a TPR above 87 percent even with very high thresholds. The KNN behavior is also stable (it manages to maintain an FPR below 1 percent and a TPR above 93 percent). This shows that both classifiers present great certainty about most of the learned data.

On the other hand, MLP achieved a TPR closer to that of Random Forest. However, MLP's FPR is closer to 3 percent (Random Forest achieves this rate with an FPR below 1 percent). The worst results were achieved with SVM: a maximum TPR close to 93 percent and FPR greater than 2.5 percent. Nonetheless, if we think of an "ideal" classifier for the commercial scenario (for example, antivirus), when a goodware would never be classified as a malware, the MLP would also exhibit this property but with a false-negative rate ($FNR = 1 - TPR$) close to 80 percent.

We compared two classification approaches that could be employed daily in industry. The first uses an incremental learning method[11]: as soon as new samples appear, we update our model with them, grouping these samples with all others that we already collected. To do so in this experiment, we divided the samples into two groups: one containing samples created up to a certain month of a year (in the case of malware, this corresponds to the year the sample was collected and sent to us from our partner; in the case of goodware, we fetched the creation date from their headers and assumed them correct), which are used for training, and the other with samples created 1 month after that said month, which are used for validation.

For example, if the month indicated is February and the year indicated is 2015, the training set contained samples that were created up until February 2015 (a cumulative set), while the validation set contained samples created only in March 2015 (1 month after February 2015). This simulates a real-world situation because we train with data that were already seen, that is, malware and goodware that were created before a particular month, and test with data from the present (from the selected month).

With both sets created, we used them in two off-the-shelf machine-learning classifiers—KNN and Random Forest—in collected samples from 2012 to 2017, excluding months that have no goodware or malware to test. Figure 3 shows the experiment results (accuracy, F1 score, recall, and precision) for KNN. The best accuracy obtained by KNN was in July 2013, with 98.81 percent, and the worst, in December 2012, with only 35.95 percent. The F1 score is also its worst in December 2012, with just 39.69 percent, reaching its peak in April 2014, with 99.29 percent. The same happens for the recall, with only 24.80 percent in December 2012 and 98.72 percent in April 2014. A different scenario occurs with precision, which achieves 100 percent about 10 times and has the worst result in December 2015, with 91 percent.

In Figure 4 we can see Random Forest results for the same experiment. The best accuracy obtained by Random Forest was in November 2016, with 99.38 percent, and the worst in December 2012, with only 15.43 percent. The F1 score is also worst in December 2012, with only 0.97 percent, reaching its peak in September 2013, with 99.40 percent. The same happened for recall, with only 0.49 percent in December 2012 and 99.06 percent in September 2013. A different scenario occurs with precision, which achieves 100 percent more than 20 times. The worst result, 95.92 percent, was in December 2015. The fact that precision is always higher than 90 percent in both cases shows that the false-positive

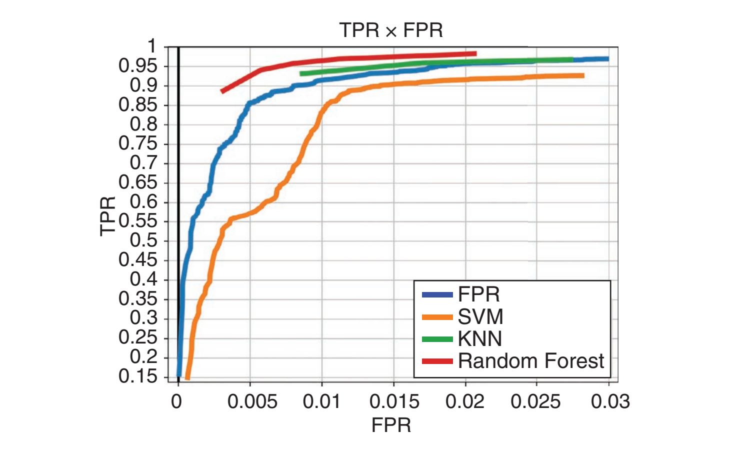

Figure 2. The ROC curve of the classifier results when using distinct thresholds.

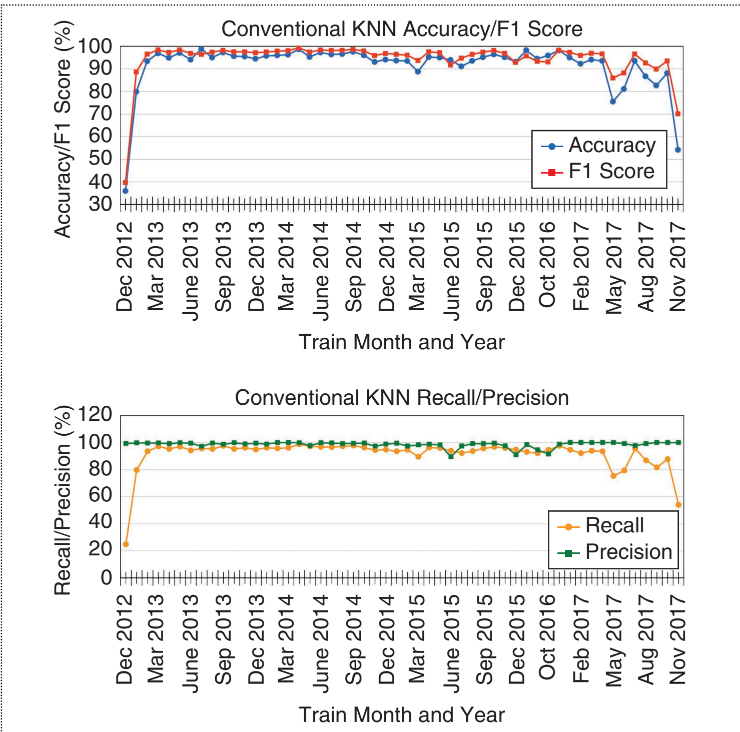

Figure 3. The results obtained in KNN when updating database samples over time.

rate is always low; that is, few examples of goodware are classified as malware.

On the other hand, both classifiers have limitations in preventing false negatives in this case, because recall never achieves 100 percent. Both classifiers had similar results in most cases. This is clear in the overall results (collecting the sum of confusion matrices from all classifiers). Random Forest achieved 93.18 percent for accuracy, 95.50 percent for F1 score, 92.46 percent for recall, and 98.74 percent for precision, while KNN

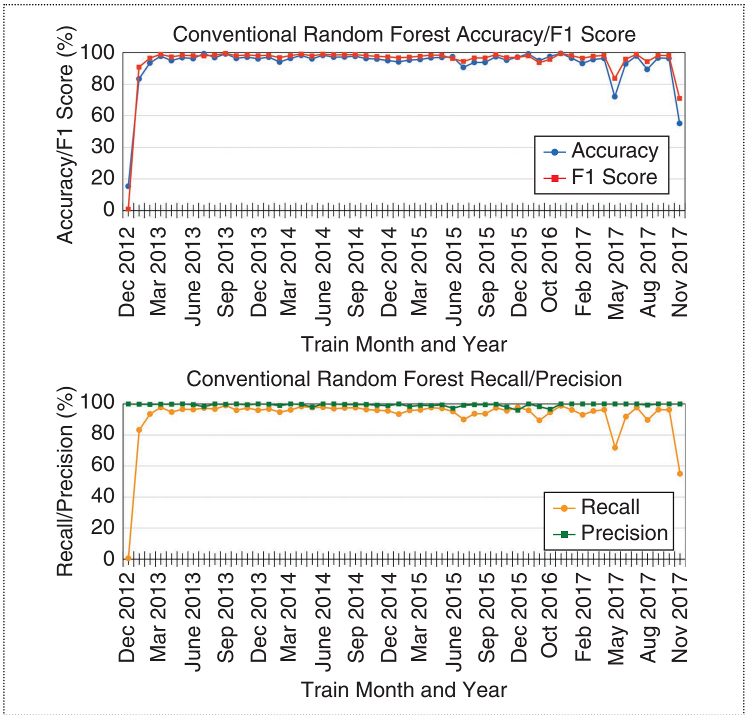

Figure 4. The results obtained in the Random Forest classifier when updating database samples over time.

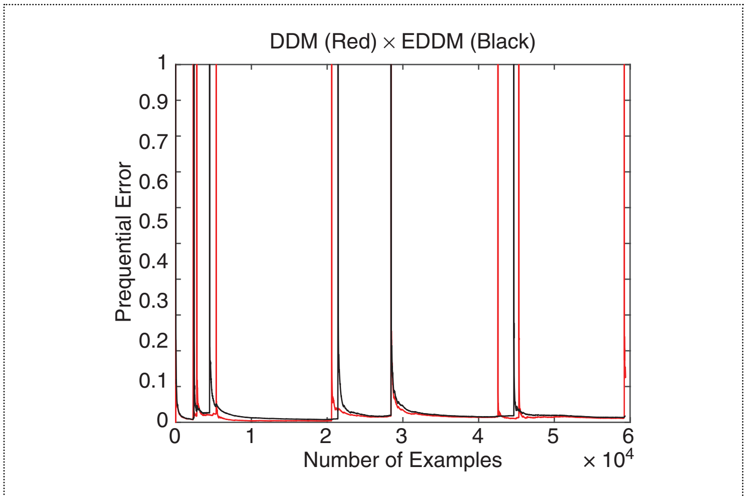

Figure 5. The DDM and EDDM detected drifts. Peaks in prequential error indicate a drift point.

achieved 93.22 percent for accuracy, 95.53 percent for F1 score, 92.41 percent for recall, and 98.86 percent for precision. The significant drop and recovery of all metrics through time in both classifiers demonstrate the existence of a concept drift in this dataset and indicate that this approach is not good for handling this problem.

In data streams, it is normal for occurring events to compromise the performance of classification systems. Such events may be concept drifts, which means that the system needs updating using the most recent examples.[11] To analyze the existence of drifts in the proposed dataset, we applied two of the most popular drift detectors found in the literature: DDM[12] and EDDM.[13] Both DDM and EDDM are online supervised methods based on sequential error (prequential) monitoring; that is, each incoming example, processed separately, estimates the prequential error rate. In this way, both methods assume that the increasing prequential error rate suggests the occurrence of concept drifts.

These drift detectors triggered two levels: warning and drift. The warning level suggests that the concept starts to drift, then an alternative classifier is updated using the examples that rely on this level. The drift level suggests that the concept drift occurred; then, the alternative classifier built during the warning level is used to replace the current classifier. In these experiments, both DDM and EDDM used Hoeffding trees as base classifiers, and our dataset was ordered by date, alternating malware and goodware [since we have fewer examples of goodware (imbalanced data), we used a circular list, repeating the first samples so that malware and goodware always alternated]. In total, 59,408 alternating samples were used (half malware and half goodware). We present the following evaluation metrics: accuracy, recall, precision, and the detection points. Because some drifts occur in the gradual transition period, we consider as true detections only the common detection points suggested by both DDM and EDDM in a range of at least 1,000 examples. Therefore, all five drifts suggested by EDDM and six of eight drifts suggested by DDM are assumed to be true detections. All detections are shown in Figure 5, where red is DDM and black is EDDM; a peak in prequential error indicates a drift point.

Before analyzing the detected drifts, it is possible to compare the overall results obtained by DDM and EDDM with the incremental learning method. DDM obtained an accuracy of 98.70 percent, an F1 score of 98.81 percent, recall of 98.87 percent, and precision of 98.75 percent, while EDDM obtained accuracy of 98.36 percent, an F1 score of 98.57 percent, recall of 98.42 percent, and precision of 98.72 percent. Comparing this to the previous approach, DDM was the best, followed by EDDM, KNN, and Random Forest, showing the need to take into account the change of concept, because both algorithms were about five percentage points better in almost all metrics.

DDM detected drift in points 2,320, 2,801, 5,353, 20,590, 28,444, 42,559, 45,320, and 59,233, while EDDM detected drift in points 2,419, 4,476, 21,436, 28,432, and 44,652. Combining these points, we created five intervals (2,320–2,419, 4,476–5,353, 20,590–21,436, 28,432–28,444, and

44,652–45,320) to compare the samples before and after the drift, as shown in Figure 6. Given a start point *S* and an endpoint *E*, both detected by DDM or EDDM, two sets were created by interval: one containing samples around point *S* (70 samples before that point and 70 after), which we call *before-drift set* (purple), and one containing samples around point *E* (also 70 samples before that point and 70 after), which we call *after-drift set* (light green). With both sets we can see what happens between the interval (from point *S* to *E*), which we call *drift window*, an interval where we assume that the transition of concept occurs.

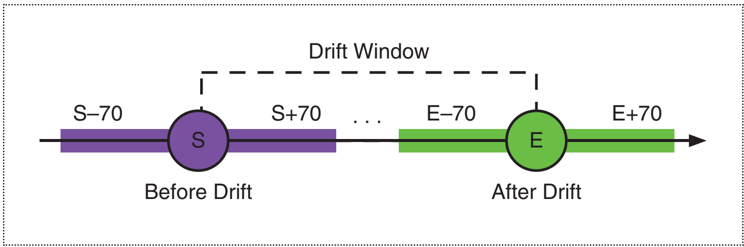

Figure 6. The drift window scheme. Samples around starting point *S* comprise before-drift set, and samples around ending point E comprise after-drift set.

An extremely useful tool for visualizing high-dimensional data is t-SNE,[14] which creates a faithful representation of these data in a lower-dimensional space (usually two-dimensional or three-dimensional), clustering similar samples. Figure 7 shows a t-SNE projection of before-drift and after-drift sets of each interval listed, where blue and green points represent goodware before and after the drift, and red and yellow points represent malware before and after a drift. Well-defined clusters are also well-defined concepts, and those clusters that are not so well defined may still be in a transition phase. From this we can conclude the following.

1) In January 2013 (2,320–2,419), a new concept of malware and goodware appears (left yellow cluster and small green cluster next to it).
2) In February 2013 (4,476–5,353), a new concept of goodware appears (left green cluster), and a small malware cluster appears in the center (yellow points next to blue and green points). Also, we can see that there are no well defined malware clusters on the right, which could indicate a transition phase.
3) In November 2013 (20,590–21,436), a new concept of malware appears [next to goodware (blue and green points) on the left], and more goodware appears next to the malware cluster in the right.
4) In April 2014 (28,432–28,444), a new concept of goodware appears in the top right, a few examples of new malware among goodware appear in bottom right, and a small cluster of malware appears toward the top.
5) In April 2015 (44,652–45,320), new concepts of goodware appear (right and bottom green clusters), and a large number of new malware examples appears in the center (yellow cluster surrounded by a few red points).

To complement the projections, we also present the distribution of malware families for before- and after-drift sets in each interval. With that, we can better understand what is happening in each drift detected, because new malware families usually imply new concepts. As shown in Figure 8, we conclude the following about each interval (in addition to previous conclusions):

1) In January 2013, the new concept of malware that appears is represented by about 28 percent of the malware after the drift in this period (from homa to dillerch families). We can also see some families before drift that do not appear after it (allaple, virut, and others).
2) In February 2013, the small malware cluster appearing in the projection is represented by the new families that have low prevalence after the drift, such as hoax, conjar, and bestafera. Malware that appear after the drift have a prevalence of only about 29 percent. Some families, such as wecod, present only before the drift (with about 18 percent prevalence in total) and simply disappear after it.
3) In November 2013, again, a small malware cluster that appears in the projection after the drift is represented by three families with low prevalence in this period (delf, amonetize, and delfinject). Also, some families disappear after the drift, such as installmonster.
4) In April 2014, the new concepts are represented by the families razy, autoit, spamies, and bancos, that is, the yellow points dispersed in the projection. Some families, such as atraps, agentb, banbra, jacard, and midia, do not appear again after the drift.
5) In April 2015, families barys, limitail, iespy, bancos, and bankfraud appear after the drift as new concepts. Although a small prevalence, together they represent that cluster in the center of the projection. The families agentb, zusy, delf, delphi, and score do not show up again.

Finally, due to the previously mentioned peculiarities of Brazil, malware targeting its users may change to other file types or go against the trend of more global malware,[15] therefore occurring in drifts unseen in other countries.

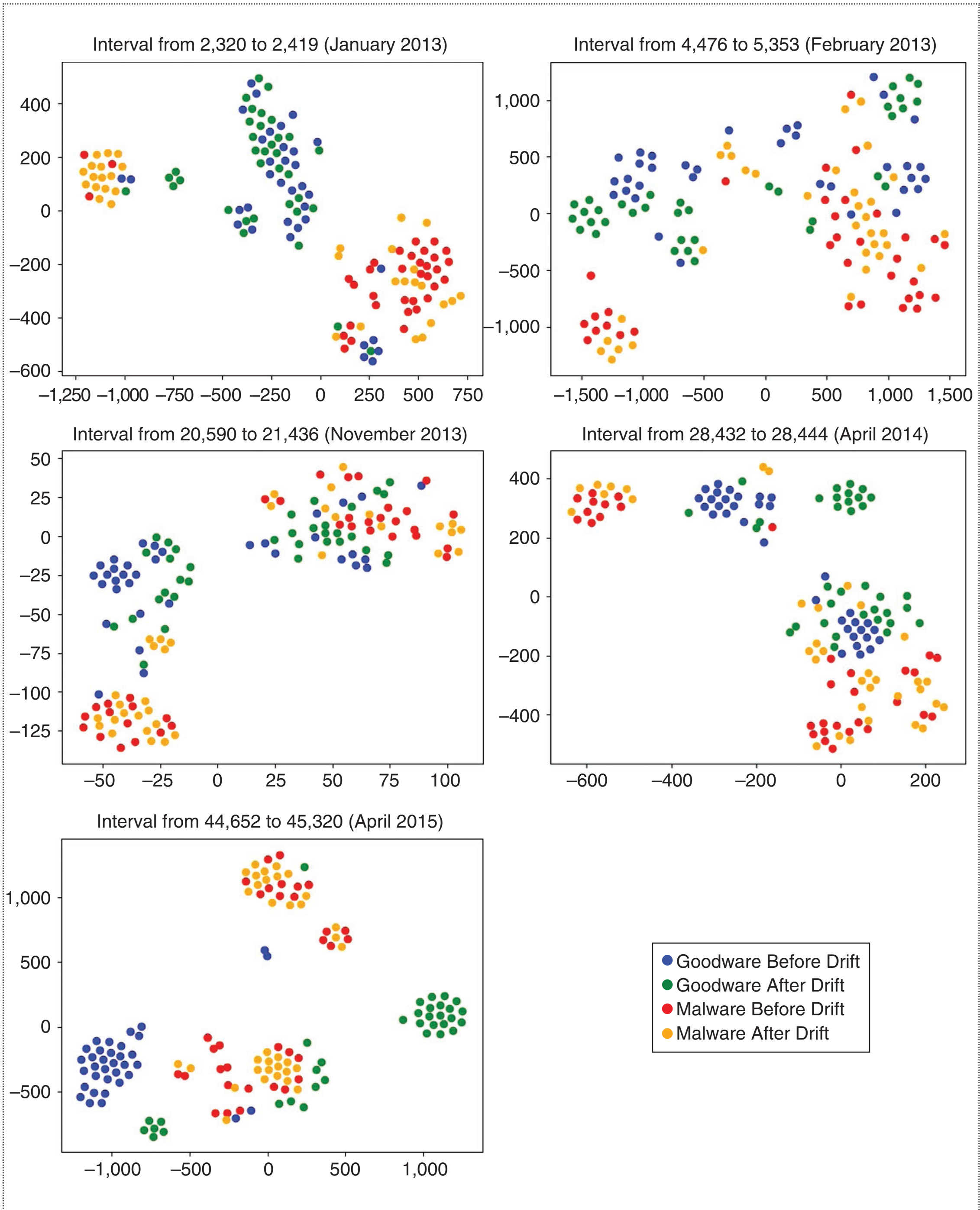

Figure 7. Projections of before-drift and after-drift sets for each interval listed.

In this article, we presented the methodology used to collect and a method to statically extract programs' attributes to produce comparable labeled feature vectors. These vectors of attributes and their corresponding description were obtained from the database we built for this article, which we will make available to the scientific community. Thus, we can highlight this article's main points:

1) Our research involved a real dataset containing 50,820 samples, where 21,116 are goodware and 29,704 are malware.

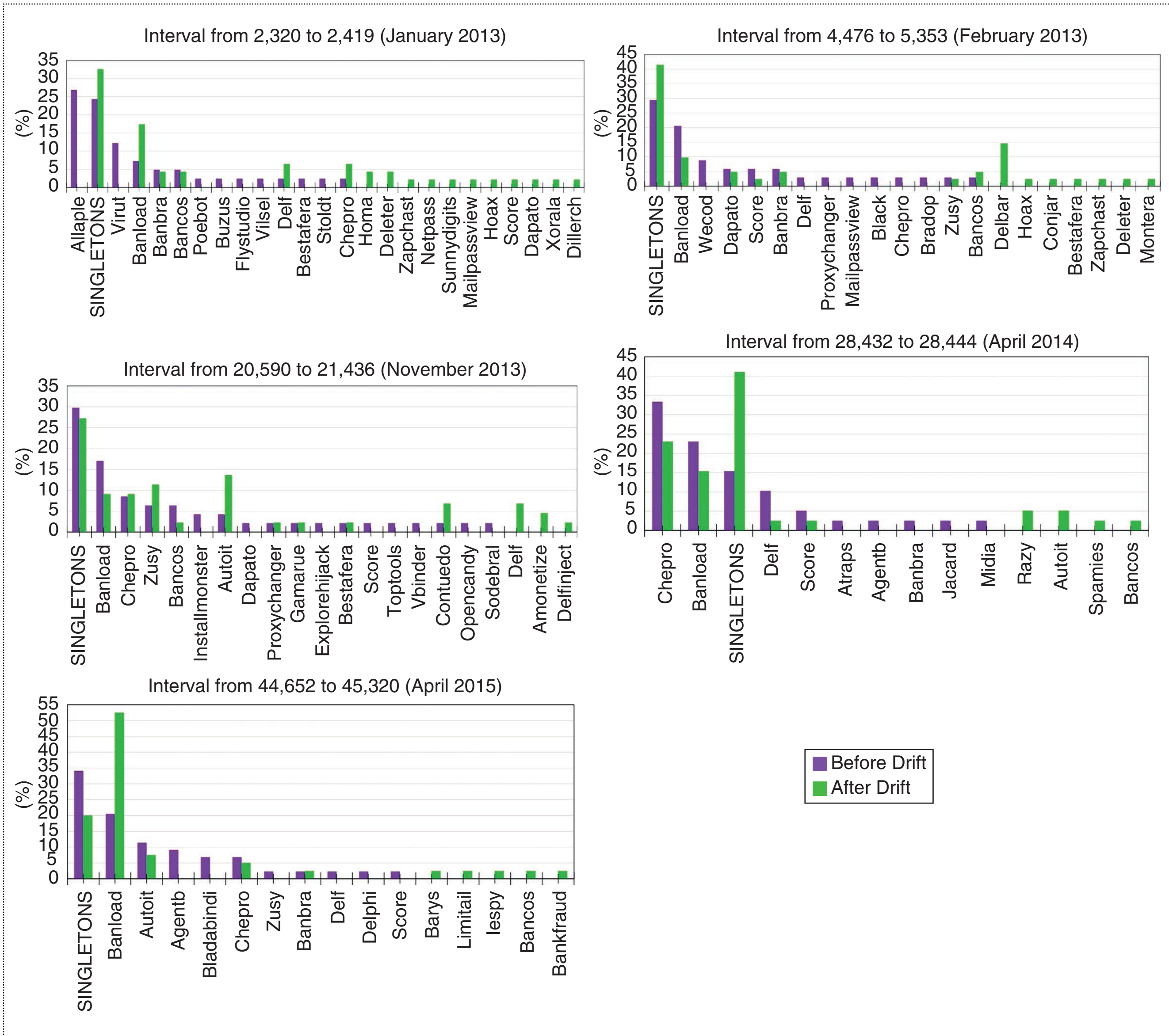

Figure 8. The distribution of malware families for before- and after-drift sets in each interval listed.

2) Traditional experiments that try to reach 100 percent accuracy are not good in this area because they do not take into account the change of concept and thus do not reflect a real scenario.
3) The industry is interested in machine learning because it reaches precision levels close to 100 percent with few false positives; that is, benign programs are rarely blocked.
4) Despite having quite good results, updating the training dataset by adding new samples only (incremental learning) is not be the best approach due to the samples' change of concept.
5) Online drift detectors, such as DDM and EDDM, help improve the classification performance as time goes by, because they obtained better overall results than incremental learning approaches.
6) The t-SNE projections presented for each interval detected, as well as their distribution of malware families before and after the proposed drift, prove the existence of concept drift in our dataset, because sample clusters form before and after it and new families appear.

In addition, because concept drift can affect the classification system's effectiveness, the main recommendation is to update it constantly, always with the most recent samples. One possible solution is to update it periodically. However, when the concept is stable, some of these updates will be unnecessary. Such updates increase the computational cost (bad for systems that receive a large amount of data). Hence, it would be ideal to detect the moment when the drift occurs and only then update the classification system (as DDM and EDDM).

We intend to study other solutions to better describe the types of concept drifts presented in this scenario, including deep learning, as well as to aggregate the extraction of dynamic features observed in the collected samples, which will enable us to use hybrid approaches and apply several other classification techniques to compare with the results gathered from static analysis. ■

Acknowledgments

We thank Google for supporting this article through the 2017 Google Research Awards for Latin America.

References

1. W. Huang and J. W. Stokes, "Mtnet: A multi-task neural network for dynamic malware classification," in *Proc. 13th Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2016, pp. 399–418.
2. O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, July 2015, pp. 1–8.
3. C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. v. Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Proc. IEEE Symp. Security and Privacy*, May 2012, pp. 65–79.
4. A. R. A. Gregio, D. S. Fernandes, V. M. Afonso, P. L. de Geus, V. F. Martins, and M. Jino, "An empirical analysis of malicious Internet banking software behavior," in *Proc. 28th Annu. ACM Symp. Applied Computing*, 2013, pp. 1830–1835.
5. V. Blue. (2014). Rsa: Brazil's 'Boleto Malware' stole nearly $4 billion in two years. *ZDNet*. [Online]. Available: https://www.zdnet.com/article/rsa-brazils-boleto-malware-stole-nearly-4-billion-in-two-years/
6. M. Pietrek. (2016). Peering inside the PE: A tour of the Win32 portable executable file format. *Microsoft*. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms809762.aspx
7. P. D. Turney and P. Pantel. (2010, Mar. 4). From frequency to meaning: Vector space models of semantics. arXiv. [Online]. Available: http://arxiv.org/abs/1003.1141
8. C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to information retrieval*. New York: Cambridge University Press, 2008.
9. A. Damien et al. (2016). Tflearn: Deep learning library featuring a higher-level API for TensorFlow. *Github*. Available: https://github.com/tflearn/tflearn, 2016.
10. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learning Res.*, vol. 12, pp. 2825–2830, 2011.
11. F. A. Pinage, E. M. dos Santos, and J. M. P. da Gama, "Classification systems in dynamic environments: An overview," *Wiley Interdisciplinary Rev.: Data Mining Knowledge Discovery*, vol. 6, no. 5, pp. 156–166, 2016.
12. J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004*, A. L. C. Bazzan and S. Labidi, Eds. Berlin: Springer Berlin Heidelberg, 2004, pp. 286–295.
13. M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo-Merino, A. Bifet, R. Gavald, and R. Morales-Bueno. 2006), Early drift detection method. *ResearchGate*, Berlin. [Online] Available: https://www.researchgate.net/publication/245999704_Early_Drift_Detection_Method
14. L. van der Maaten and G. Hinton, "Visualizing high-dimensional data using t-SNE," *J. Mach. Learning Res.*, vol. 9, pp. 2579–2605, 2008.
15. M. Botacin, P. L. de Geus, and A. R. A. Gregio, "The other guys: Automated analysis of marginalized malware," *J. Computer Virology Hacking Techniques*, vol. 14, no. 1, pp. 87–98, 2018. [Online]. Available: https://doi.org/10.1007/s11416-017-0292-8.

Fabrício Ceschin is a Ph.D. student at Federal University of Paraná, Brazil, where he received his M.S. degree in informatics. His research interests include machine learning and deep learning applied to security. He has received support from the Google Research Awards for the Latin America program. Contact him at fjoceschin@inf.ufpr.br.

Felipe Pinagé is a postdoctoral researcher at the Federal University of Paraná, Brazil. His research interests include machine learning, data streams, and concept drift. Pinagé received a Ph.D. in informatics from the Federal University of Amazonas, Brazil, in 2017. Contact him at fapinage@inf.ufpr.br.

Marcos Castilho is a full professor at the Federal University of Paraná, Brazil. His research interests include artificial intelligence and free software for scientific computing. He coordinates research and development projects funded by the Brazilian Ministry of Education, with a focus on informatics applied to education, transparency, and monitoring of public resources. Castilho received a Ph.D. in informatics from the Université Toulouse III Paul

Sabatier, France, in 1998. Contact him at marcos@inf.ufpr.br.

David Menotti is an associate professor at the Federal University of Paraná, Brazil. He has research projects funded by the Brazilian National Counsel of Technological and Scientific Development (CNPq) and had an approval for a Ph.D. IBM fellowship nomination in 2017. His research interests include pattern recognition, computer vision, image processing, and machine learning. He has been a CNPq research fellow since 2015 and an IEEE Senior Member since March 2018. Contact him at menotti@inf.ufpr.br.

Luiz S. Oliveira is an associate professor at the Federal University of Paraná, Brazil. His research interests include pattern recognition, machine learning, and computer vision. In 2017, he received an IBM Faculty Award for work on texture classification. He has been a Brazilian National Counsel of Technological and Scientific Development research fellow (level 1D) since 2006. Contact him at luiz.oliveira@ufpr.br.

André Grégio is an assistant professor at the Federal University of Paraná, Brazil. His research interests include several aspects of computer and network security, such as countermeasures against malicious codes, security data visualization/analysis, and mobile security. He has research projects funded by the Brazilian National Counsel of Technological and Scientific Development, the Brazilian Ministry of Health, and Google. Grégio received a Ph.D. in computer engineering from the University of Campinas, Brazil, in 2012. Contact him at gregio@inf.ufpr.br.